



**EDB**

Postgres® for the AI Generation

## **Lakehouse analytics**

リリース *January 28, 2025*

**EnterpriseDB Corporation.**

2025 年 01 月 29 日



# 目次

|              |                                                                   |           |
|--------------|-------------------------------------------------------------------|-----------|
| <b>第 1 章</b> | <b>Introduction</b>                                               | <b>1</b>  |
| 1.1          | Postgres の迅速な分析                                                   | 1         |
| 1.2          | Postgres ネイティブ                                                    | 1         |
| 1.3          | ベクトル化された実行                                                        | 1         |
| 1.4          | カラムナ型ストレージ                                                        | 2         |
| 1.5          | Lakehouse Sync                                                    | 2         |
| 1.6          | フルマネージドサービス                                                       | 2         |
| 1.7          | 今日試してみる                                                           | 2         |
| <b>第 2 章</b> | <b>Concepts - EDB Postgres Lakehouse</b>                          | <b>3</b>  |
| 2.1          | 主要な概念                                                             | 3         |
| 2.1.1        | Lakehouse ノード                                                     | 3         |
| 2.1.2        | Lakehouse テーブル                                                    | 4         |
| 2.1.3        | 管理対象ストレージの場所                                                      | 4         |
| 2.1.4        | Lakehouse Sync                                                    | 4         |
| 2.2          | 用語                                                                | 4         |
| 2.2.1        | レイクハウス                                                            | 4         |
| 2.2.2        | ベクトル化されたクエリエンジン                                                   | 5         |
| 2.2.3        | デルタテーブル                                                           | 5         |
| 2.3          | その仕組み                                                             | 5         |
| 2.3.1        | レベル 100                                                           | 5         |
| 2.3.2        | レベル 200                                                           | 6         |
| 2.3.3        | レベル 300                                                           | 6         |
| <b>第 3 章</b> | <b>Quick Start - EDB Postgres Lakehouse</b>                       | <b>9</b>  |
| 3.1          | はじめに                                                              | 9         |
| 3.1.1        | 知っておくべきこと                                                         | 9         |
| 3.1.2        | 基本アーキテクチャ                                                         | 10        |
| 3.2          | はじめに                                                              | 10        |
| 3.2.1        | Lakehouse ノードを作成する                                                | 10        |
| 3.3          | Lakehouse ノードの動作                                                  | 11        |
| 3.3.1        | ノードに接続します                                                         | 11        |
| 3.3.2        | コンストレインを理解する                                                      | 12        |
| 3.4          | ベンチマークデータセットを検査する                                                 | 12        |
| 3.5          | ベンチマークデータセットを照会する                                                 | 13        |
| <b>第 4 章</b> | <b>Querying Delta Lake Tables in S3-compatible object storage</b> | <b>15</b> |
| 4.1          | 概要                                                                | 15        |
| 4.2          | 前提条件                                                              | 15        |
| 4.3          | 外部テーブルの作成                                                         | 16        |

|              |                                               |           |
|--------------|-----------------------------------------------|-----------|
| 4.4          | 外部テーブルのクエリー                                   | 16        |
| <b>第 5 章</b> | <b>Lakehouse Sync</b>                         | <b>17</b> |
| 5.1          | 概要                                            | 17        |
| 5.2          | Lakehouse Sync の実行                            | 17        |
| 5.2.1        | 前提条件                                          | 17        |
| 5.2.2        | Lakehouse Sync に移動します                         | 17        |
| 5.2.3        | Lakehouse Sync を定義する                          | 17        |
| 5.2.4        | テーブルの選択                                       | 18        |
| 5.2.5        | Lakehouse Sync を開始する                          | 18        |
| <b>第 6 章</b> | <b>Reference - EDB Postgres® AI Lakehouse</b> | <b>19</b> |
| 6.1          | Supported cloud providers and regions         | 19        |
| 6.2          | Supported AWS instances                       | 20        |
| 6.3          | Benchmarking datasets                         | 20        |
| 6.4          | Queries                                       | 21        |
| 6.5          | Delta Lake Table tools                        | 21        |
| 6.5.1        | Delta Lake テーブルの作成                            | 21        |
| 6.6          | PGAA functions reference                      | 22        |
| 6.6.1        | pgaa.create_storage_location                  | 22        |
| 6.7          | DirectScan, fallback modes, and EXPLAIN       | 23        |
| 6.8          | User management                               | 24        |
| 6.9          | Loading data (sync or bring your own)         | 24        |
| 6.9.1        | Lakehouse 同期を使用したデータのロード                      | 24        |
| 6.9.2        | 独自のデータの持ち込み                                   | 25        |

# 第1章 Introduction

EDB Postgres Lakehouse は、ベクトル化されたクエリエンジンを追加し、ストレージをコンピューティングから分離することにより、Postgres の機能を分析ワークロードに拡張します。データレイクハウスの構築がこれまでにない簡単になりました。Postgres を使用するだけです。

## 1.1 Postgres の迅速な分析

Postgres Lakehouse は、EDB Postgres<sup>®</sup> AI プラットフォームのコア製品であり、Postgres ユーザーが愛するシンプルさと使いやすさを維持しながら、オブジェクトストレージの列状データの分析クエリをサポートするように Postgres を拡張します。

Postgres Lakehouse を使用すると、列型データに対するベクトル化クエリ実行用に最適化された Postgres を搭載した一時的なスケルトンゼロコンピューティングリソースである Lakehouse ノードを使用して Postgres データを照会できます。

## 1.2 Postgres ネイティブ

Postgres エコシステムを決して離れないでください。

Postgres Lakehouse ノードは、EDB Advanced Server または EDB Postgres ExtendedPGE のいずれかを Postgres エンジンとして実行し、分析用のデータは、オープンソースの Delta Lake プロトコルを使用してオブジェクトストレージに列状テーブルとして保存されます。

EDB Postgres Lakehouse は「単なる Postgres」です。任意の Postgres クライアントで照会でき、すべての Postgres クエリ、ファンクション、ステートメントを完全にサポートしているため、既存のクエリを変更したり、ビジネスインテリジェンスソフトウェアを再構成したりする必要はありません。

## 1.3 ベクトル化された実行

Postgres Lakehouse は、Apache DataFusion のベクトル化 SQL クエリエンジンを使用して、ネイティブ Postgres と比較して 5~100 倍平均 30 倍高速に分析クエリを実行しながら、必要に応じてネイティブ実行にフォールバックします。

## 1.4 カラムナ型ストレージ

Postgres Lakehouse は、オブジェクトストレージの「レイクハウステーブル」を照会するように最適化されており、オープンソースデータベースの機能をオープンテーブル形式に拡張します。現在、Delta Lake プロトコルに従って保存された「デルタテーブル」の照会をサポートしています。

## 1.5 Lakehouse Sync

トランザクションソース最初には EDB Postgres® AI Cloud Service データベースのテーブルから独自のデータを、ストレージロケーション最初には S3 オブジェクトストレージの管理対象ロケーションのレイクハウステーブルに同期できます。

## 1.6 フルマネージドサービス

EDB Postgres AI Cloud Service 旧 EDB BigAnimal を使用して、Postgres Lakehouse ノードを起動できます。いくつかのデルタテーブルを含むストレージバケットをレイクハウスノードでポイントし、トランザクション Postgres データベースで同じデータを照会する場合よりも短い時間で分析 OLAP クエリの結果を取得します。

Postgres Lakehouse ノードは、EDB Postgres AI - AWS でホストされた環境を使用しているお客様が利用できるようになりました、すぐに追加のクラウド環境にロールアウトされます。

## 1.7 今日試してみる

Postgres Lakehouse の使用を開始するのは簡単です。5 分でレイクハウスノードをプロビジョニングし、TPC-H、TPC-DS、クリックベンチ、10 億行チャレンジなどのプリロードされたベンチマークデータのクエリを開始します。

## 第2章 Concepts - EDB Postgres Lakehouse

EDB Postgres Lakehouse は、EDB Postgres® AI プラットフォーム上の運用データに対して Rapid Analytics を実行するためのソリューションです。

### 2.1 主要な概念

- レイクハウスノード は、 管理対象ストレージの場所 の레이크ハウステーブル を照会します。
- **Lakehouse Sync** は、ソースデータベースのトランザクションテーブル から **Lakehouse** テーブルを作成できます。

それがどのように収まるかは次のとおりです。

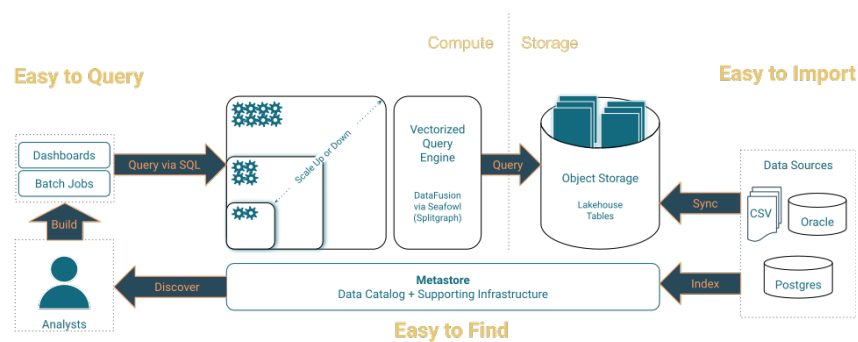


図 1: Level 50 basic architecture

#### 2.1.1 Lakehouse ノード

Postgres Lakehouse ノードは Postgres で、Lakehouse テーブルを照会するように最適化されたベクトル化クエリエンジンを備えていますが、完全な互換性を得るために Postgres にフォールバックします。

Lakehouse ノードはステートレスで一時的なものです。ワークロード要件に基づいて、それらをスケールアップまたはダウンします。

## 2.1.2 Lakehouse テーブル

Lakehouse テーブルは、分析用に最適化され、アナリティクスエコシステムの残りの部分と相互運用可能な、圧縮性の高い列型ストレージ形式を使用して保存されます。現在、Postgres Lakehouse ノードは、デルタプロトコルを使用して保存されたテーブル「デルタテーブル」を読み取ることができ、Lakehouse Sync はそれらを書き込むことができます。

## 2.1.3 管理対象ストレージの場所

**管理対象ストレージ場所**は、オブジェクトストレージで Lakehouse テーブルを編成できる場所で、Postgres Lakehouse がそれらを照会できるようにします。

管理対象ストレージの場所は、ファイルレイアウトを制御し、お客様に代わってレイクハウステーブルを書き込むオブジェクトストレージ内の場所です。技術的には、これらをバケットに保存するのは実装の詳細です。これは実際には、独自のバケットを持ってくる「外部ストレージロケーション」もサポートする今後の「ストレージロケーション」機能のサブセットです。

## 2.1.4 Lakehouse Sync

Lakehouse Sync は、EDB Postgres AI プラットフォームの一部として提供されるデータ移行サービスです。トランザクションデータベースから宛先のストレージ場所にあるレイクハウステーブルにテーブルを「同期」できます。現在、EDB Postgres AI Cloud Service 旧 BigAnimal として知られているソースデータベースをサポートしています。将来的には、より多くのソースデータベース Oracle などと展開場所オンプレミスデータベースをサポートする予定です。Debezium を使用してビルドされています。

## 2.2 用語

### 2.2.1 レイクハウス

「[Lakehouse architecture](#)」は、データエンジニアリングの実践であり、「Data Lake」と「Data Warehouse」を組み合わせたもので、両方の利点を提供します。アーキテクチャの中心的な教義は、データがオブジェクトストレージに保存されるということです。一般的に Parquet のような列指向形式で、さまざまなクエリエンジンが特定のクエリに最適なコンピューティングリソースを使用して、独自の特殊な目的にそれ进行处理できます。

## 2.2.2 ベクトル化されたクエリエンジン

ベクトル化されたクエリエンジンは、列指向データでクエリを実行するように最適化されたクエリエンジンです。ほとんどの分析エンジンは、ベクトル化されたクエリ実行を使用します。Postgres Lakehouse は [Apache DataFusion](#) を使用します。

## 2.2.3 デルタテーブル

私たちは、特定の形式へのオーバーコミットを避けるために「レイクハウステーブル」という用語を使用していました例えば、最終的に Iceberg または Hudi をサポートする可能性があるため。しかし、技術的には、[Delta Tables](#) を使用しています。デルタテーブルは、「Delta Lake」仕様とオープンプロトコルに従って、Parquet ファイルと JSON メタデータの明確に定義されたコンテナです。Delta Lake は、Linux Foundation プロジェクトです。

## 2.3 その仕組み

Postgres Lakehouse は、多くのテクノロジーを使用して構築されています。

- PostgreSQL
- [Seafowl](#)、Splitgraph から取得した分析データベース
- [Apache DataFusion](#)、Seafowl が使用するクエリエンジン
- [Delta Lake](#) および具体的には [delta-rs](#) デルタテーブルのストレージおよび取得レイヤーを実装するため

### 2.3.1 レベル 100

Postgres Lakehouse について理解する最も重要な点は、ストレージをコンピューティングから分離することです。これにより、それらを独立してスケーリングできます。これは、クエリが予測不能でスパイクになる可能性がある分析ワークロードに最適です。接続されたハードドライブにデータを保持するためだけに、マシンをほとんどアイドル状態にしておいたくないでしょう。代わりに、データをオブジェクトストレージおよび高圧縮可能な形式で保持し、必要に応じて照会に必要なコンピューティングのみをプロビジョニングできます。

コンピューティング側では、ベクトル化されたクエリエンジンはレイクハウステーブルを照会するように最適化されていますが、完全な互換性を得るために Postgres にフォールバックします。

ストレージ側では、レイクハウステーブルは、アナリティクス用に最適化された圧縮性の高い列型ストレージ形式を使用して保存されます。

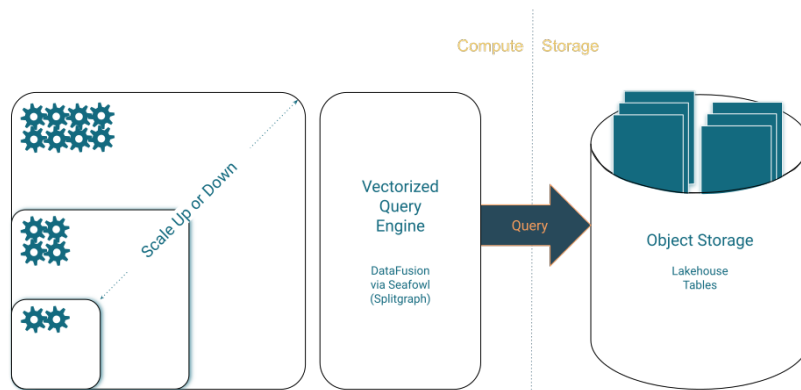


図 2: Level 100 Architecture

### 2.3.2 レベル 200

次に、これらのサービスがどのように連携するかを示すもう少し包括的な図を示します。

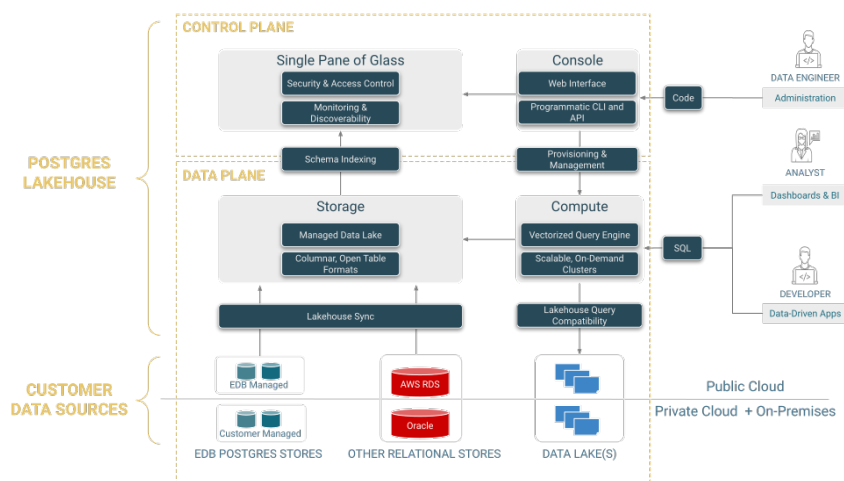


図 3: Level 200 Architecture

### 2.3.3 レベル 300

これは、「ボックスの内容」のより詳細な拡大ビューです。

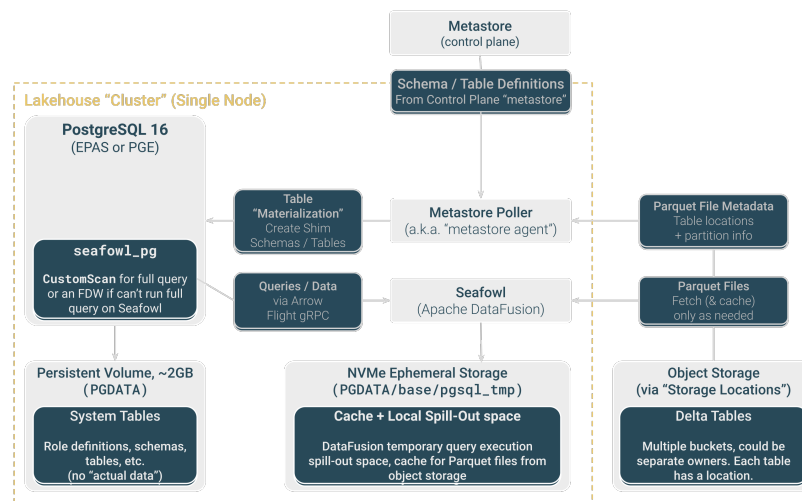


図 4: Level 300 Architecture



## 第3章 Quick Start - EDB Postgres Lakehouse

このガイドでは、次のことを行います。

1. Lakehouse ノードを作成する
2. 優先する Postgres クライアントを使用してノードに接続します
3. オブジェクトストレージ内のサンプルデータ TPC-H、TPC-DS、Clickbench、または 1BRC を照会します

詳細および高度なユースケースについては、[PGAA functions reference](#) を参照してください。

### 3.1 はじめに

Postgres Lakehouse は、新しいタイプの Postgres 「クラスター」 実際には単なる 1 つのノードであり、EDB Postgres® AI Cloud Services 旧「BigAnimal」としてプロビジョニングできます。これには、オブジェクトストレージ内の列状の「レイクハウステーブル」を高速クエリーするためのベクトル化クエリエンジン Apache DataFusion ベース Delta Lake protocol を使用します。

#### 3.1.1 知っておくべきこと

詳細については、[Concepts - EDB Postgres Lakehouse](#) および [PGAA functions reference](#) を参照してください。ただし、これはその要点をカバーするはずです。

- Postgres Lakehouse ノードは一時的ですが、ローカルキャッシュとスピルアウト領域用の NVMe ハードドライブを備えています。現在、AWS の EDB ホストリージョンへの展開をサポートしています。
- Postgres Lakehouse ノードは、オブジェクトストレージ内のベンチマークデータセットを照会するように事前構成されています。
- Postgres Lakehouse は、通常「読み取り専用」ですが、それ以外の場合は、すべての有効な Postgres クエリーをサポートします。
- システムテーブルは書き込み可能なため、ユーザーとロールを作成できます。これらのテーブルは、接続されたブロックストレージに保存されるため、インスタンスを再起動するまで持続し、バックアップ/復元操作の一部として保存されます。それ以外の場合、Lakehouse テーブルは最終的にオブジェクトストレージに保存されるため、バックアップの一部になりません。

### 3.1.2 基本アーキテクチャ

Lakehouse ノードのボックスの内容は次のとおりです。

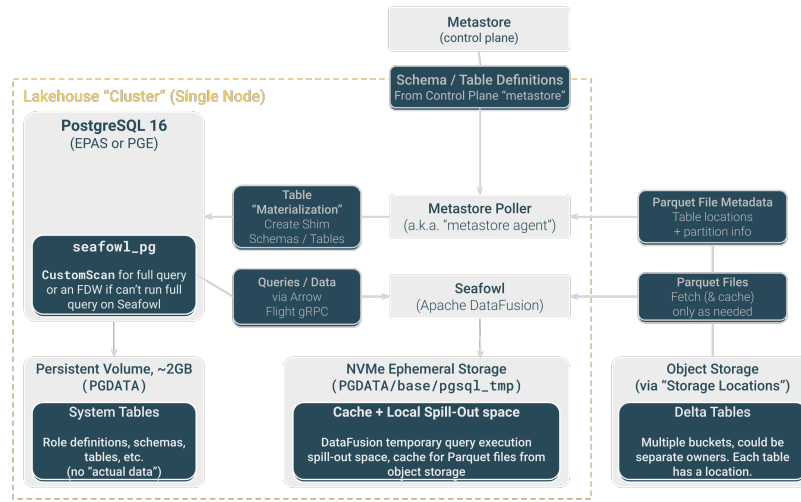


図 1: Level 300 Architecture

## 3.2 はじめに

EDB Postgres AI アカウントが必要です。ログインしてプロジェクトを作成したら、クラスターを作成できます。

### 3.2.1 Lakehouse ノードを作成する

プロジェクトページの「Create New」ドロップダウンの下に「Lakehouse Analytics」オプションが表示されます。

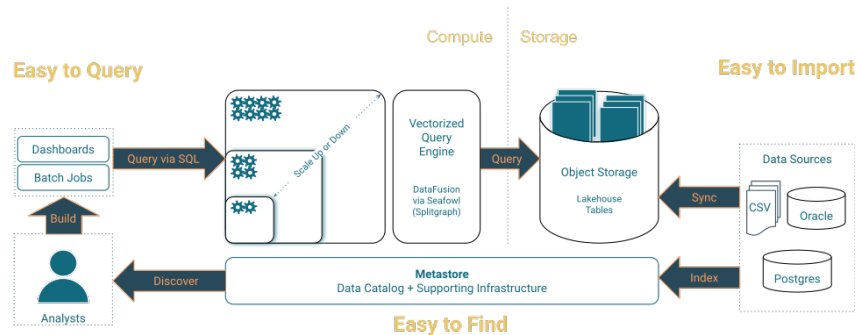


図 2: Level 50 basic architecture

[Lakehouse Analytics] ボタンを選択すると、次のような構成ウィザードが起動します。

クラスターのプロビジョニングには 10~15 分かかる場合があります。

図 3: Create Lakehouse Node Wizard Step 1

### いくつかの注意事項

- 独自のパスワードを提供する必要があります。安全なパスワードを作成し、パスワードマネージャーに保存します。私たちの側では保存しません。
- BigAnimal Hosted リージョンでは、AWS のみがサポートされています。
- EDB Postgres Advanced Server または EDB Postgres ExtendedPGE を選択できます。価格計算ツールは、通常の EPAS または PGE シングルノードクラスターの場合と同様に、ライセンスコストと同じ式を使用します。
- NVMe ドライブを備えたインスタンスのみがサポートされています。これは M6ID ファミリーです。NVMe ドライブは、クエリの処理中にスピルアウト領域に使用されます。NVMe ドライブ上のすべてのデータは、クラスターがシャットダウンすると失われます。システムテーブルユーザー、ロールなどの永続データは、接続されたブロックストレージデバイスに保存され、再起動またはバックアップ/復元サイクル後も存続します。
- Postgres 16 のみがサポートされています。

サポートされているインスタンスサイズに関する注意事項については、[Reference - Supported AWS instances](#) を参照してください。

## 3.3 Lakehouse ノードの動作

### 3.3.1 ノードに接続します

EDB Postgres AI Cloud Service 旧 BigAnimal から他のクラスターに接続するのと同じ方法で、Postgres クライアントを使用して Lakehouse ノードに接続できます。クラスターの詳細ページに移動し、接続文字列をコピーします。

たとえば、`.pgpass blob` を `~/.pgpass` にコピーします。`$YOUR_PASSWORD` を、クラスター起動時に指定したパスワードに置き換えてください。次に、接続文字列をコピーし、`psql` または `pgcli` への引数として使用できます。

通常、Postgres クライアントでデータベースに接続できる必要があります。私たちはすべてのイントロスペクションクエリーが機能することを期待しています。そうでないクエリーが見つかった場合、それはバグです。

### 3.3.2 コンストレインを理解する

- すべてのクラスターは EPAS または PGE を使用します。したがって、接続したときにインストール内のこれらのフレーバーからボイラープレートテーブルが表示されることが予想されます。
- クエリ可能データベンチマークデータセットのようなは、デルタテーブルとしてオブジェクトストレージに保存されます。すべてのクラスターは、1~1000 のスケールファクターで TPC-H、TPC-DS、クリックベンチ内のベンチマークデータを含むストレージバケットをポイントするようにプリロードされています。
- 現時点では AWS のみがサポートされています。Bring Your Own Account BYOA はサポートされていません。
- EDB Postgres AI アカウントでアクティブ化されている任意のリージョンにクラスターを展開できます。各リージョンには、ベンチマークデータのコピーを含むバケットがあるため、クラスターを起動すると、それに最も近い場所のベンチマークデータを使用します。
- クラスターは一時的なものです。システムテーブルのデータを除き、データはハードドライブに保存されません。ロール、ユーザー、および付与。クラスターを再起動するか、クラスターをバックアップして復元すると、これらのシステムテーブルが復元されます。ただし、オブジェクトストレージのデータは変更されないままです。
- クラスターは、オブジェクトストレージ内のデータの読み取り専用クエリーをサポートしますが、ユーザーなどを作成するためのシステムテーブルへの書き込みクエリーをサポートします。オブジェクトストレージに直接書き込むことはできません。新しいテーブルを作成することはできません。
- 独自のデータをオブジェクトストレージにロードする場合は、[Reference - Bring your own data](#) を参照してください。

## 3.4 ベンチマークデータセットを検査する

ベンチマークデータセットを検査します。すべてのクラスターには、すぐに使用できるベンチマークデータがあります。pgcli を使用している場合、`\dn` を実行して、使用可能なテーブルを表示できます。

使用可能なベンチマークデータセットは次のとおりです。

- TPC-H、スケールファクター 1、10、100、および 1000
- TPC-DS、スケールファクター 1、10、100、および 1000
- クリックベンチ、1 億行
- 10 億行チャレンジ

ベンチマークデータセットの詳細については、[Reference - Available Benchmarking datasets](#)(`./reference/datasets`) を参照してください。

### 3.5 ベンチマークデータセットを照会する

いくつかの基本的なクエリーを実行してみることができます。

```
edb_admin> select count(*) from clickbench.hits;
+-----+
| count |
+-----+
| 99997497 |
+-----+
SELECT 1
Time: 0.945s

edb_admin> select count(*) from brc_1b.measurements;
+-----+
| count |
+-----+
| 10000000000 |
+-----+
SELECT 1
Time: 0.651s
```

注 `search_path` は使用しないでください。 [PGAA functions reference](#) を読む

その他の注意事項と構文/クエリの互換性に関する情報のページ。



## 第4章 Querying Delta Lake Tables in S3-compatible object storage

### 4.1 概要

外部テーブルを使用すると、SQL を使用して S3 互換のオブジェクトストレージに保存されているデータにアクセスおよび照会できます。S3 互換オブジェクトストレージのデータを参照する外部テーブルを作成し、標準の SQL コマンドを使用してデータを照会できます。

### 4.2 前提条件

- EDB Postgres AI アカウントと Lakehouse ノード。
- デルタレイクテーブルとして保存されたデータを含む S3 互換のオブジェクト保存場所。\*データの準備方法の詳細については、[独自のデータの持ち込み](#) を参照してください。
- パブリックバケットでない限り、S3 互換のオブジェクト保存場所にアクセスするための資格情報。\*これらの資格情報はデータベース内に保存されます。この目的のために、権限を制限した別のユーザーを作成することをお勧めします。
- クロスリージョンのレイテンシーのため遅くなる
- データ転送には AWS コスト 0.01~0.02/GB が発生します。現在、これらの下りコストは渡されませんが、当社はそれらを追跡し、インスタンスを終了する権利を留保します。

最初のステップは、データが存在する S3 互換オブジェクトストレージを参照する外部ストレージ場所を作成することです。保存場所は、データにアクセスするために参照するデータベース内のオブジェクトです。各保存場所には、この目的のための名前があります。

名前付けの保存場所の作成は、SQL で `pgaa.create_storage_location` ファンクションを実行して実行されます。`pgaa` は、外部ストレージの場所を照会する機能を提供する拡張機能と名前空間の名前です。`create_storage_location` ファンクションは、新しい保存場所の名前と、S3 互換のオブジェクト保存場所の URI をパラメーターとして受け取ります。ファンクションは、オプションで 3 番目のパラメーター `options` を取ることができます。これは、オプション設定を指定するための JSON オブジェクトであり、[PGAA functions reference](#) で詳細に説明されています。たとえば、オプションで、保存場所のアクセスキー ID とシークレットアクセスキーを指定して、プライベートバケットへのアクセスを有効にできます。

次の例では、パブリック S3 互換オブジェクト保存場所を参照する外部テーブルを作成します。

```
SELECT pgaa.create_storage_location(  
  name => sample-data,
```

(次のページに続く)

(前のページからの続き)

```
url => s3://pgaa-sample-data-eu-west-1,
options => {},
msl_id => NULL
);
```

次の例では、プライベート S3 互換のオブジェクト保存場所を参照する外部ストレージ場所を作成します。

```
SELECT pgaa.create_storage_location(
  name => private-data,
  url => s3://my-private-bucket,
  options => {"access_key_id": "my-access-key-id", "secret_access_key": "my-secret-
→access-key"},
  msl_id => NULL
);
```

## 4.3 外部テーブルの作成

外部格納場所を作成した後、格納場所のデータを参照する外部テーブルを作成できます。次の例では、S3 互換のオブジェクト格納場所にデルタレイクテーブルを参照する外部テーブルを作成します。

```
CREATE TABLE public.customer () USING PGAA WITH (pgaa.storage_location = sample-data,
→pgaa.path = tpch_sf_1/customer);
```

CREATE TABLE ステートメントではスキーマが定義されていないことに注意してください。pgaa 拡張機能は、保存場所にスキーマが定義されることを想定しており、スキーマ自身自身は、pgaa.path オプションで指定されたパスに保存されているスキーマから派生します。pgaa 拡張機能は、デルタテーブルの列に最適な Postgres 同等のデータ型を推論します。

## 4.4 外部テーブルのクエリー

外部テーブルを作成した後、標準の SQL コマンドを使用して外部テーブルのデータを照会できます。次の例では、前の手順で作成した外部テーブルを照会します。

```
SELECT COUNT(*) FROM public.customer;
```

## 第5章 Lakehouse Sync

### 5.1 概要

Lakehouse Sync の実行は、ある時点でトランザクションデータベースから情報をキャプチャし、その情報を管理対象店舗ロケーション MSL に同期する方法です。

Lakehouse 同期プロセスは、トランザクションデータベースデータを MSL に保存される Lakehouse テーブルに編成します。このプロセスにより、Lakehouse ノードでデータを照会できます。これは、Lakehouse テーブル用に設計されたベクトル化クエリエンジンを使用して、よりパフォーマンスの高いクエリ用に最適化されています。

### 5.2 Lakehouse Sync の実行

#### 5.2.1 前提条件

- EDB Postgres AI<sup>®</sup> Cloud Service によってホストおよび管理される Postgres クラスター

#### 5.2.2 Lakehouse Sync に移動します

1. EDB Postgres AI Console に移動します。
2. ランディングページから、同期するデータベースインスタンスを含むプロジェクトを選択します。ランディングページに表示されない場合は、**Projects** セクションの **View Projects** リンクを選択し、そこからプロジェクトを選択します。
3. 左側のナビゲーションバーで **Migrate** ドロップダウンを選択し、**Migrations** を選択します。
4. **Create New Migration** ボタンを選択します。

#### 5.2.3 Lakehouse Sync を定義する

5. 同期に **名前** を付けて、同期する **ソースクラスター** と **データベース** を選択します。
6. 使用する MSL を既に作成している場合は、使用可能な MSL のリストからその MSL を選択し、以下の **テーブルの選択** に進みます。まだの場合は、**Create New Management Storage Location** ボタンを選択して、**Add Management Storage Location** ダイアログを開きます。
7. 新しい MSL の AWS リージョンを選択します。

8. [管理対象ストレージの場所の追加] ダイアログの下部にあるフォームに場所プレフィックスを設定して、MSL の定義を完了します。ロケーションプレフィックスは、MSL に関連付けられているリソースとアセットを識別するために使用される一意の名前です。

**Add Managed Storage Location**

Step 1: Select a region below | Step 2: Define a location prefix | Step 3: Create the storage location

| Provider                  | Region         | Deployment                     | Storage Locations |
|---------------------------|----------------|--------------------------------|-------------------|
| <input type="radio"/> AWS | AP South 1     | EDB Postgres® AI Cloud Service | 0                 |
| <input type="radio"/> AWS | AP SouthEast 2 | EDB Postgres® AI Cloud Service | 0                 |
| <input type="radio"/> AWS | EU Central 1   | EDB Postgres® AI Cloud Service | 0                 |
| <input type="radio"/> AWS | EU West 1      | EDB Postgres® AI Cloud Service | 1                 |
| <input type="radio"/> AWS | EU West 2      | EDB Postgres® AI Cloud Service | 0                 |

Region: No region selected | Prefix:

Cancel | Create Managed Storage Location

図 1: List of MSLs

9. **Create Management Storage Location** ボタンを選択します。

## 5.2.4 テーブルの選択

10. ページの上部にある **Get Started** タブの横にある **Tables** タブを選択し、移行に含めるテーブルと列を選択します。

## 5.2.5 Lakehouse Sync を開始する

11. **Start Lakehouse Sync** ボタンを選択します。
12. 成功した場合、レイクハウスの同期は、[移行] ページの [最近の] 移行の下に [作成中] ステータスと表示されます。同期の実行にかかる時間は、同期されるデータの量によって異なり、数時間かかる場合があります。

**注釈:** プロジェクトの最初の同期には、必要なインフラストラクチャのプロビジョニングのため、数時間かかります。

## 第6章 Reference - EDB Postgres® AI Lakehouse

EDB Postgres® AI Lakehouse は初期の製品です。最終的には、複数のクラウドとオンプレミスにわたる展開モードをサポートする予定です。ただし、現在、展開できる場所とクエリできるデータの点でかなり制限されています。

Lakehouse で最高のエクスペリエンスを得るには、[Quick Start - EDB Postgres Lakehouse](#) に従ってベンチマークデータを照会する必要があります。次に、Lakehouse Sync で独自のデータをロードしてみるか、独自のデータを持ち込んで *external tables* を使用できます。ご興味がありましたら、私たちに連絡してください。あなたのユースケースと潜在的な機会について詳しくお話しします。

このセクションでは、知っておくべき重要な事項のいくつかについて詳しく説明します。

### 6.1 Supported cloud providers and regions

**AWS のみ**現在、すべてのレイクハウス機能レイクハウスノード、管理対象ストレージロケーション、およびレイクハウス同期のサポートは AWS に制限されています。

**EDB ホスト型のみ** “Bring Your Own Account” BYOA リージョンは、現在 Lakehouse リソースでサポートされていません。サポートは、AWS 上の **EDB Postgres® AI Hosted** 環境別名「EDB-Hosted AWS リージョン」のみに制限されています。

これは、次のリージョンのいずれかから選択できることを意味します。

- 北米 **米国東部 1** 米国東部 2 \* 米国西部 2
- ヨーロッパ **EU 中部 1** EU 西部 1 \* EU 西部 2
- アジア \* AP South 1
- オーストラリア \* AP SouthEast 2

正確には

- Lakehouse ノードは、EDB ホスト型 AWS リージョンでのみプロビジョニングできます
- 管理対象ストレージロケーションは、EDB ホスト型 AWS リージョンでのみ作成できます
- Lakehouse Sync は、EDB ホスト型 AWS リージョンのソースデータベースからのみ同期できます

これらの制限は、製品の改良を続けるにつれて削除されます。最終的に、私たちは BYOA、Azure、GCP をすべての Lakehouse のユースケースでサポートする予定です。

## 6.2 Supported AWS instances

Lakehouse ノードを展開するときは、`m6id` インスタンスファミリーからインスタンスタイプを選択する必要があります。重要なのは、これらのインスタンスには、NVMe ドライブが接続されていることです。

インスタンスは一時的なものです。これらの NVMe ドライブは、クエリの処理中に\*スピルアウト領域にのみ使用され、およびディスクにデルタテーブルをキャッシュするために使用されます。クラスターがシャットダウンすると、NVMe ドライブ上のすべてのデータが失われます。

システムテーブルは永続化されます。システムテーブルの永続データユーザー、ロール、\*などは、接続されたブロックストレージデバイスに保存され、一時停止/再開サイクル後も存続します。

サポートされているインスタンス

| API Name             | Memory    | vCPUs     | Cores | Storage                        |
|----------------------|-----------|-----------|-------|--------------------------------|
| <i>m6id.large</i>    | 8.0 GiB   | 2 vCPUs   | 1     | 118 GB NVMe SSD                |
| <i>m6id.xlarge</i>   | 16.0 GiB  | 4 vCPUs   | 2     | 237 GB NVMe SSD                |
| <i>m6id.2xlarge</i>  | 32.0 GiB  | 8 vCPUs   | 4     | 474 GB NVMe SSD                |
| <i>m6id.4xlarge</i>  | 64.0 GiB  | 16 vCPUs  | 8     | 950 GB NVMe SSD                |
| <i>m6id.8xlarge</i>  | 128.0 GiB | 32 vCPUs  | 16    | 1900 GB NVMe SSD               |
| <i>m6id.12xlarge</i> | 192.0 GiB | 48 vCPUs  | 24    | 2850 GB (2 * 1425 GB NVMe SSD) |
| <i>m6id.16xlarge</i> | 256.0 GiB | 64 vCPUs  | 32    | 3800 GB (2 * 1900 GB NVMe SSD) |
| <i>m6id.24xlarge</i> | 384.0 GiB | 96 vCPUs  | 48    | 5700 GB (4 * 1425 GB NVMe SSD) |
| <i>m6id.32xlarge</i> | 512.0 GiB | 128 vCPUs | 64    | 7600 GB (4 * 1900 GB NVMe SSD) |

## 6.3 Benchmarking datasets

Lakehouse ノードをプロビジョニングすると、サンプルベンチマークデータセットを含む同じリージョンのパブリック S3 バケットをポイントするように事前構成されます。

スキーマ名で参照することにより、これらのデータセットのテーブルを照会できます。

| Schema Name          | Dataset                      |
|----------------------|------------------------------|
| <i>tpcds_sf_1</i>    | TPC-DS, Scale Factor 1       |
| <i>tpcds_sf_10</i>   | TPC-DS, Scale Factor 10      |
| <i>tpcds_sf_100</i>  | TPC-DS, Scale Factor 100     |
| <i>tpcds_sf_1000</i> | TPC-DS, Scale Factor 1000    |
| <i>tpch_sf_1</i>     | TPC-H, Scale Factor 1        |
| <i>tpch_sf_10</i>    | TPC-H, Scale Factor 10       |
| <i>tpch_sf_100</i>   | TPC-H, Scale Factor 100      |
| <i>tpch_sf_1000</i>  | TPC-H, Scale Factor 1000     |
| <i>clickbench</i>    | ClickBench, 100 million rows |
| <i>brc_1b</i>        | Billion row challenge        |

!!!note ClickBench データに関する注意事項

データ列 EventData は、日付ではなく整数です。

ClickBench 列名には大文字が含まれるため、引用符で囲む必要がありますが、Postgres の引用符のない識別子は小文字と大文字が区別されません。例

```
❌ select "Title" from clickbench.hits;
```

```
❌ select Title from clickbench.hits; !!!
```

## 6.4 Queries

通常、**READ ONLY** クエリーがサポートされています。オブジェクトストレージに直接書き込むことはできません。これには、すべての Postgres ビルトインファンクション、ステートメント、およびタイプが含まれます。また、展開するディストリビューションに応じて、EPAS または PGE が提供するものも含まれます。

通常、データを挿入、更新、削除、または変更することはできません。CREATE TABLE は使用できますが、ノードに通常の Postgres テーブルを作成するためにのみ、ノードは一時的なもので、終了すると破棄されることに注意してください。CREATE TABLE... USING PGAA を使用して、S3 互換オブジェクトストレージのデルタテーブルを参照する外部テーブルを作成することもできます。独自の ETL スクリプトまたは Lakehouse Sync を使用して、アウトオブバンドでデータをバケットにロードする必要があります。詳細については、[Loading data \(sync or bring your own\)](#) を参照してください。

1 つの例外は、ユーザー、ロール、付与を保存するために使用される Postgres システムテーブルです。これらのテーブルは、バックアップとリストアに含まれるローカルブロックデバイスに保存されます。したがって、CREATE USER または CREATE ROLE または GRANT USAGE ができます。これらのユーザー/ロール/付与は、再起動と復元後も残ります。

## 6.5 Delta Lake Table tools

### 6.5.1 Delta Lake テーブルの作成

**deltalake Python** ライブラリの使用

**deltalake** Python ライブラリを使用して、デルタテーブルを作成し、バケットに書き込むことができます

## lakehouse-loader ユーティリティの使用

EDB がこのタスク用に作成した `lakehouse-loader` ユーティリティを使用して、任意の Postgres インスタンスからストレージバケットのレイクハウステーブルにデータをエクスポートすることもできます。

例、`lakehouse-loader` ユーティリティを使用する場合

```
export PGPASSWORD="..."
export AWS_ACCESS_KEY_ID="..."
export AWS_SECRET_ACCESS_KEY="..."

# export other AWS envvars

./lakehouse-loader postgres-to-delta postgres://test-user@localhost:5432/test-db -q
→ "SELECT * FROM some_table" s3://my-bucket/my_schema/my_table
```

これにより、`test-db` データベースの `some_table` テーブルから `my-bucket` バケットの `my_schema/my_table` パスのデルタテーブルにデータがエクスポートされます。

`my_schema/my_table` パスのデルタテーブルを参照する外部テーブルを作成することにより、Lakehouse ノードでこのテーブルを照会できるようになりました。その方法の詳細については、[External Tables](#) を参照してください。

## 6.6 PGAA functions reference

### 6.6.1 pgaa.create\_storage\_location

#### 概要

S3 互換のオブジェクト保存場所を参照する新しい保存場所を作成します。

#### パラメーター

| Parameter      | Type        | Description                                                                        |
|----------------|-------------|------------------------------------------------------------------------------------|
| <i>name</i>    | <i>text</i> | 保存場所の名前                                                                            |
| <i>uri</i>     | <i>text</i> | S3 互換のオブジェクト保存場所の URI <code>s3://bucket-name`または`s3://bucket-name/prefix`など</code> |
| <i>options</i> | <i>json</i> | 保存場所のオプション設定                                                                       |

## オプション

| Option                         | Type                 | Description                  |
|--------------------------------|----------------------|------------------------------|
| <code>access_key_id</code>     | <code>text</code>    | 保存場所のアクセスキー ID               |
| <code>secret_access_key</code> | <code>text</code>    | 保存場所のシークレットアクセスキー            |
| <code>session_token</code>     | <code>text</code>    | 保存場所のセッショントークン               |
| <code>region</code>            | <code>text</code>    | 保存場所のリージョン                   |
| <code>endpoint</code>          | <code>text</code>    | 保存場所のエンドポイント                 |
| <code>bucket</code>            | <code>text</code>    | 保存場所のバケット                    |
| <code>use_https</code>         | <code>boolean</code> | 保存場所に HTTPS の代わりに HTTP を使用する |
| <code>skip_signature</code>    | <code>boolean</code> | 保存場所の署名検証をスキップする             |

## 6.7 DirectScan, fallback modes, and EXPLAIN

Postgres Lakehouse は、クエリ全体を DataFusion、可能な場合にクエリを処理するために使用されるベクトル化クエリに「プッシュダウン」できる場合に最速です。将来的には、部分的なプッシュダウンのサポートを追加するため、これはより細かくなります。

Postgres Lakehouse は、2つのモードでクエリを実行できます。最初に、Seafoal DataFusion に基づいた専用の列型データベースを使用してクエリ全体を実行しようとします。たとえば、JSON のような PostgreSQL 固有の操作を使用するために Seafoal がクエリ全体を実行できない場合、Postgres Lakehouse は PostgreSQL エグゼキュータの使用にフォールバックし、Seafoal は完全なテーブルコンテンツをストリーミングします。

クエリが非常に遅い場合、それが起こっている可能性があります。

クエリーで EXPLAIN を実行し、最上位のクエリーノードが DirectScan であることを確認することにより、使用されているモードを確認できます。例

```
explain select count from (select count(*) from tpch_sf_1.lineitem);
```

QUERY PLAN

```

-----
→ -----
→ -----
→ -----
Aggregate (cost=167.52..167.55 rows=1 width=8)
  → Append (cost=0.00..165.01 rows=1001 width=0)
    → CompatScan on "16529" lineitem_2 (cost=100.00..150.00 rows=1000 width=0)
        SeafoalPlan: logical_plan
          TableScan: tpch_sf_1.lineitem projection=[l_orderkey, l_partkey, l_
→suppkey, l_linenum, l_quantity, l_extendedprice, l_discount, l_tax, l_returnflag,
l_linestatus, l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_shipmode, l_
→comment]
(6 rows)
```

この場合、クエリーは PostgreSQL によって実行され、Seafoal はテーブルをスキャンする場合にのみ関係します下部の `CompatScan` を参照してください。この場合の修正は、内側の `COUNT(*)` 列に明示的に名前を付けることです。Seafoal は暗黙の名前 `count(*)` を付けるのに対し、PostgreSQL は `count` と呼ぶためです。

```
edb_admin=> explain select count from (select count(*) as count from tpch_sf_1.
→lineitem);
```

#### QUERY PLAN

```
-----
DirectScan: logical_plan
```

```
  Projection:  COUNT(*) AS count
```

```
    Aggregate:  groupBy=[[]], aggr=[[COUNT(UInt8(1)) AS COUNT(*)]]
```

```
      TableScan:  tpch_sf_1.lineitem projection=[]
```

```
(4 rows)
```

ここでは、上部に `DirectScan` が表示されます。これは、Seafoal がクエリ全体を実行していることを意味します。

Seafoal で完全に実行できるようにクエリを変更するのに問題がある場合は、サポートチケットをオープンしてください。

## 6.8 User management

Lakehouse ノードをプロビジョニングするときは、パスワードを入力する必要があります。このパスワードは保存しません。edb\_admin ユーザーとしてログインするために必要です。これはスーパーユーザーアカウントではありませんが、ユーザー、ロール、付与を作成する機能があります。したがって、edb\_admin 自分自身の資格情報を共有するか、他のユーザーを作成して配布できます。

## 6.9 Loading data (sync or bring your own)

### 6.9.1 Lakehouse 同期を使用したデータのロード

EDB Postgres AI Cloud Service で実行されているトランザクションデータベースがある場合は、このデータベースから管理対象ストレージの場所にテーブルを同期できます。詳細については、[How to lakehouse sync](#) を参照してください。

## 6.9.2 独自のデータの持ち込み

内部にデルタテーブルを含む任意の S3 バケットをレイクハウスノードに向けることが可能です。ただし、これにはいくつかの重要な注意事項があります最終的に解決される予定です。

### 注意事項

- テーブルは場所内に [Delta Lake Tables](#) として保存する必要があります。
- 「Delta Lake Table」または「Delta Table」は、Parquet ファイルといくつかの JSON メタデータのフォルダーです。

### バケットへのデータのロード

`lakehouse-loader` ユーティリティを使用して、任意の Postgres インスタンスからストレージバケットのデルタテーブルにデータをエクスポートできます。そのユーティリティを取得して使用方法の詳細については、[Delta Lake Table tools](#) を参照してください。

詳細については、[External Tables](#) ドキュメントを参照してください。